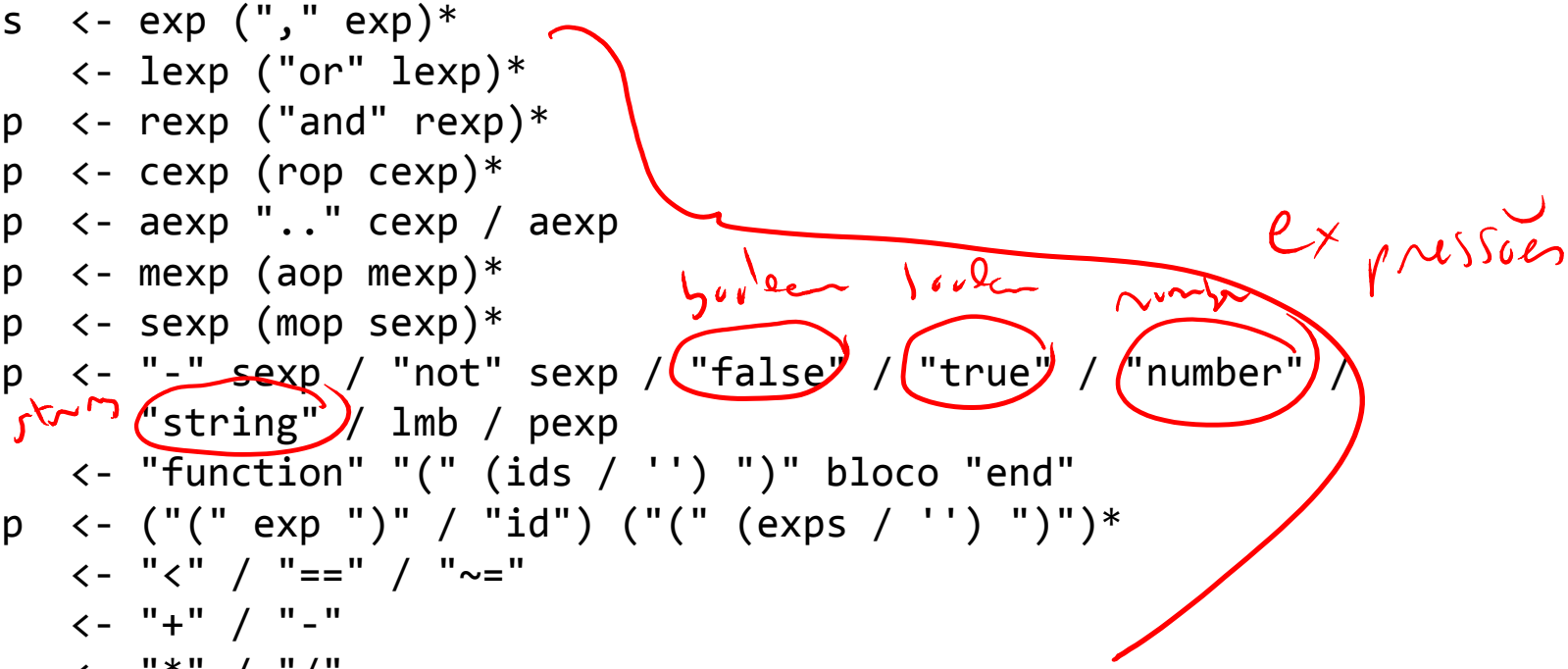
Compiladores II

Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/comp2

SmallLua - sintaxe

```
bloco <- stat* (ret / '')
stat  <- "while" exp "do" bloco "end" / "local" "id" "=" exp /
         "id" "=" exp / "function" "id" "(" (ids / '') ")" bloco "end" /
         "if" exp "then" bloco ("else" bloco / '') "end" /
         pexp
ret   <- "return" exp
ids   <- "id" ("," "id")*
exps  <- exp ("," exp)*
exp   <- lexp ("or" lexp)*
lexp  <- rexp ("and" rexp)*
rexp  <- cexp (rop cexp)*
cexp  <- aexp ".." cexp / aexp
aexp  <- mexp (aop mexp)*
mexp  <- sexp (mop sexp)*
sexp  <- "-" sexp / "not" sexp / "false" / "true" / "number" /
         "string" / lmb / pexp
lmb   <- "function" "(" (ids / '') ")" bloco "end"
pexp  <- ("(" exp ")" / "id") ("(" (exps / '') ")")*
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```



Unificação - Variáveis

$\forall a,b.\ (a, (a \to b) \to string) \Downarrow (X, (X) \to Y) \to string$

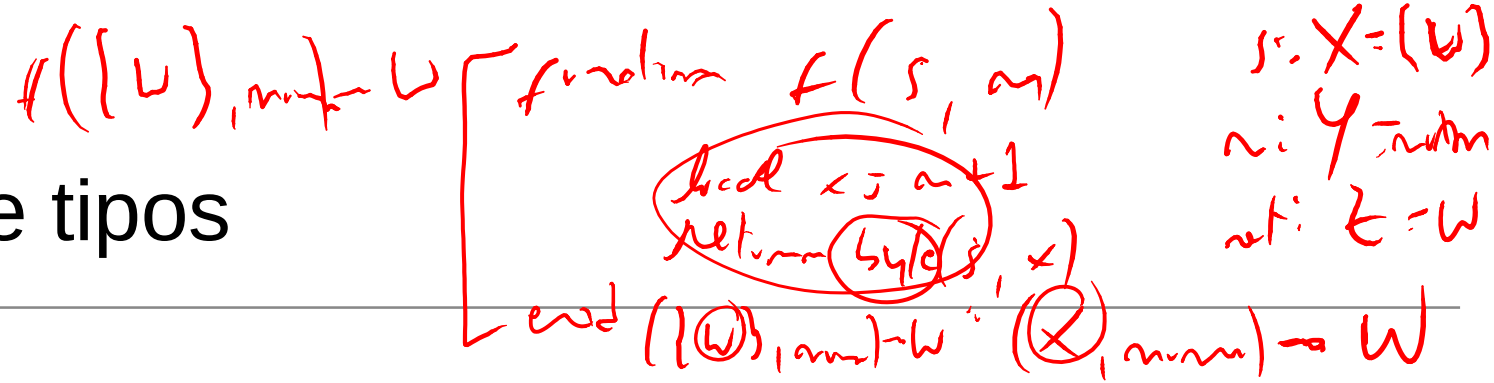
- A unificação é um algoritmo fácil de definir quando trocamos nossos parâmetros de tipo por *variáveis de tipo*
- Uma variável inicialmente não está associada a nenhum tipo, mas pode vir a ficar durante o processo de unificação
- Uma vez associada, ela não pode mais mudar; efetivamente ela assume aquele tipo
- Duas (ou mais) variáveis não associadas podem se unir: quando uma for associada, a outra também fica associada ao mesmo tipo

$(X, (X) \to Y) \to string$ $(number, (number) \to string) \to string$

Unificação - pseudocódigo

```
function unify(t1, t2):
    t1 = prune(t1)
    t2 = prune(t2)
    case t1, t2:
        match var, _:
            if occurs(t1, t2):
                error("ciclo")
            else:
                t1.type = t2
        match _, var: unify(t2, t1)
        match base, base:
            if t1.tag ~= t2.tag:
                error("incompatible")
        match func(params1, ret1), func(params2, ret2):
            if #args1 ~= #args2:
                error("arity")
            zip(unify, params1, params2)
            unify(ret1, ret2)
        match seq(elem1), seq(elem2):
            unify(elem1, elem2)
        otherwise: error("incompatible")
```

Inferência de tipos



- A unificação consegue achar uma associação entre variáveis e tipos que torna dois tipos iguais, caso ela exista

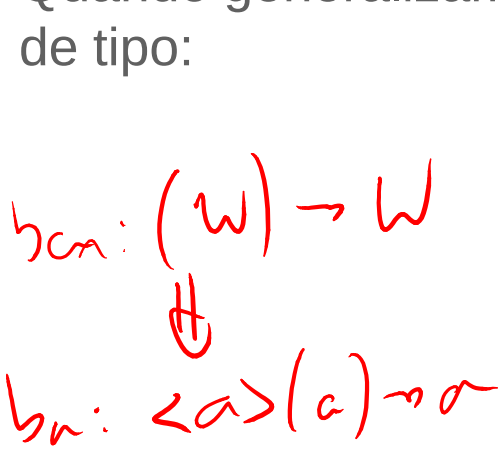
- Caso os dois tipos não tenham nenhuma variável em aberto, a unificação faz o mesmo trabalho da igualdade

- Podemos trocar todas as comparações de tipos que fazemos em nosso verificador por unificações, e permitir que o programador omita declarações de tipos

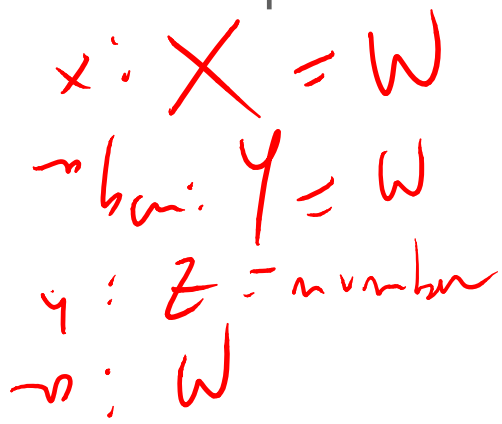
- Uma declaração omitida é substituída por uma variável em aberto, e a unificação vai dizer qual tipo essa declaração deveria ter

Inferência de tipos - generalização

- Quando generalizamos um tipo, uma variável em aberto é como um parâmetro de tipo:



```
function bar(x)
    local baz = function (y)
                    return x
                end
    return baz(1)
end
```



- Quando generalizamos `bar`, o tipo dela tem uma variável em aberto, que foi associado à variável x e por unificação foi parar no tipo de retorno

- A generalização transforma deve transformar essas variáveis de tipo em parâmetros de tipo genéricos

Inferência de tipos - especialização

- A especialização de tipos agora pode deixar variáveis de tipo em aberto, que vão seguir no processo de inferência, permitindo inferir tipos como o da função abaixo:

```
function primeiro(s)
    return byte(s, 1)
end
```

- Também podemos inferir tipos mais específicos:

```
function foo(s)
    return byte(s, 1) + 0
end
```